演習 3 (7/16)

31010 佐藤秀明

概要

- Abstract watermarking
  - プログラムを抽象的な領域Ｌに落とし込んで解釈する
  - Ｌ上で値の不変な変数＝埋め込まれた情報
  - 効率・攻撃耐性など
- まとめ
  - Software watermarking の現状・可能性

操作的意味論

- 計算機のある状態を、PC の値 c とメモリの内容 m の組 s = (c, m) で表す
- プログラム P は s から s' への遷移規則 t: s-->s' の集合で表現される
- 初期状態の集合 S_0 から始めて t に関する反射的推移的閉包 R を求めると、R は状態遷移に対する不動点になっている
  - この計算は停止する

Abstract domain とGalois connection

- 前頁で定めた状態集合 Σ を抽象的な領域 L に落とし込むことができると仮定する
- (Σ, ⊆) と (L, ≦) について、2 者の間に相互に順序保存写像が定義できるとする (Galois connection)
- Σ の不動点 R に対応して L の上限 R' が存在すれば、R' を求める計算は停止する

Abstract watermarking の概要

1. 透かし s を $s_1, s_2, \ldots, s_l$ に分割する
2. プログラム P に $s_1, s_2, \ldots, s_l$ を埋め込んでプログラム P' を作成する
3. P' を抽象的な領域 $L_1, L_2, \ldots, L_l$(≒mod 空間) に落とし込み、それぞれの領域における P' の解析結果から $s_1, s_2, \ldots, s_l$ を復元する
4. $s_1, s_2, \ldots, s_l$ を s に合成する

再び中国の剰余定理

- 中国の剰余定理 (一般化バージョン)

  $n_1, n_2, \ldots, n_l \left(< m\right)$を互いに素な自然数とする。
  自然数$c\left(< n_1 n_2 \ldots n_l\right)$に対し、自然数の組
  $\left(c_1, c_2, \ldots, c_l\right) \in [0, n_1 - 1] \times [0, n_2 - 1] \times \ldots \times [0, n_l - 1]$を用い、

  $$c = \sum_{i=1}^{l} n_1 \ldots n_{i-1} c_i n_{i+1} \ldots n_l \quad \left(mod\ n_1 \ldots n_l\right)$$

  と表すことができる。このとき、cと$\left(c_1, c_2, \ldots, c_l\right)$は
  同型写像で結ばれる。

- $(n_1, \ldots, n_l)$ が既知であれば、c と $(c_1, \ldots, c_l)$ の間の相互変換を一意に行うことができる

透かしの埋め込み (1)

- ある透かし c_i について埋め込まれるコードは以下の 3 部分に分類される
  1. 新しい変数を宣言する部分
  2. 1. の変数を初期化する部分
  3. 2. 変数を繰り返し更新する部分 (ループ内に挿入)

```
/* (1) */
int w;
/* (2)  ただし P(1) = c_i mod n_i */
w = P(1);
/* (3)  ただし Q(c_i) = c_i mod n_i */
for(;;){
  w = Q(w);
}
```

透かしの埋め込み (2)

- 例 (P(x) 、Q(x) が 2 次式のとき)
  - $P(x)=x^2+k_1x+k_0$ について、各係数の設定や埋め込むコードの例は以下のとおり

$$k_0 = -(1+c_i)+r_1 n_i$$
$$k_1 = c_i + r_2 n_i$$
$$(r_1, r_2 \text{はランダム})$$

```
w = 1;
t = w + k1;
t = w * t;
w = t + k0;
```

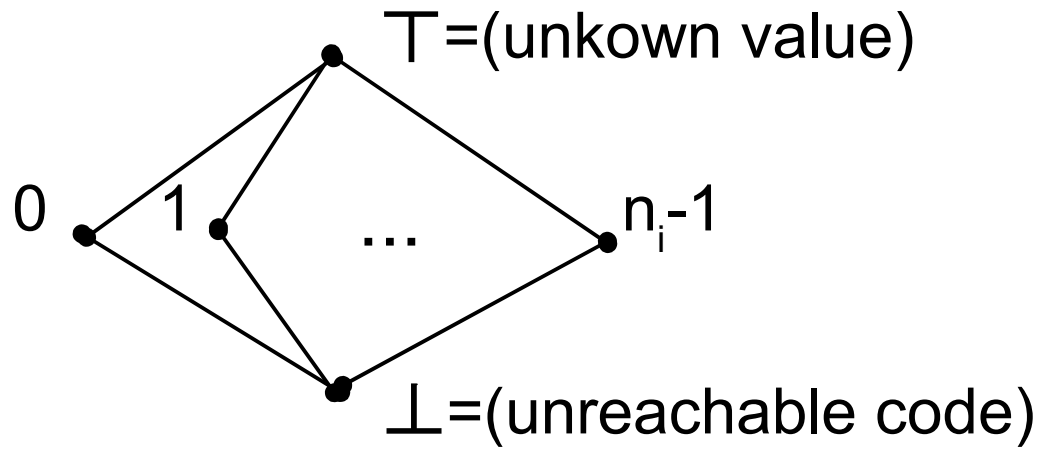
  - $Q(x)=px^2+qx+r$ について、各係数の設定や埋め込むコードの例は以下のとおり

$$r = c_i - (q\,c_i + p\,c_i^{\,2})$$
$$(p, q \text{はランダム})$$

```
t = w * p;
t = t + q;
t = t * w;
w = t + r;
```

透かしの復元(1)

- 抽象的な領域 $L_{i,x}$ は、各状態でのある変数 x の値 v を n_i で法をとった値 v' にすべて置き換えてできたものだと考える
  - ⊥から始めて不動点を求める反復作業の中で、異なる状態で異なる v' の値が現れた場合は、それらの上限を⊤と判断して停止する



透かしの復元 (2)

- 前頁の $L_{i,x}$ を、各変数 x についての直積をとった L_i へと拡張する
- 不動点が求まった後、T になっていないような ($=n_i$ で法をとった場合に定数になるような) 変数の値を c_i として復元する
  - 各 n_i が十分な大きさを持つ互いに素な自然数であれば、c_i はほぼ確実に復元できる

利点

- 互いに素な自然数の組 $(n_1, \ldots, n_l)$ を知らなければ、透かし $(c_1, \ldots, c_l)$ を計算することすら難しい
- mod 空間で計算を行うので桁溢れを無視できる
- 互いに素な素数を使い切ってしまわない限り、大量の透かし情報を何度でも埋め込むことができる
  - 重ねて透かしを埋め込んでも、もとから埋め込まれていた透かしは復元できる
- 透かしの復元に際して実行時の情報を必要としない
- コード挿入攻撃が効かない

改良

- 透かし埋め込み後に情報を復元できるかどうかをすぐ確認する
  - もとからあった定数は無視するようにする
  - 予期しない定数が復元されてしまったらキー n_i を代える
- 他の方法との併用
  - 難読化…コード改変の防止
  - Opaque predicate　…真偽の判定が難しい述語の導入
  - 定数伝搬などを用い、改変されてしまったプログラムからも何とかして透かしを回復しようと努力

評価 (論文より)

- Java に対して Java を用いて実装
- 効率…まあまあよい
  - 透かしの埋め込み・回復にかかる時間はだいたいコンパイル時間と同じくらい
  - コード自体の実行効率はほとんど変わらない
- 信頼性…まあまあよい
  - 種々の obfuscator が繰り出す攻撃が透かし情報を傷つけることはできなかった
    - コード改変を防止するため、透かしを埋め込んだ直後に自前で obfuscation をかけるのはよい方法

課題

- 透かし情報を保持する変数がとる値にはある種の特徴がある
  - mod 空間ではなくもとの整数空間で観察すると、その値は異常に大きく振れている
  - 変数値の更新履歴の列についてその階差をとると、それらの最大公約数はキー n_i の倍数になっている
- もしかしたらばれるかも…

発展

- mod 空間以外の抽象化の方法が考案されれば、より強い watermarking が可能になるかもしれない

演習 3 のまとめ (1)

- 題材 : Software watermarking
  - ソフトウェアに透かし情報を埋め込むことで知的財産の不正な使用を抑止する (cf. 不可能にする)
    - 不正にコピーした製品に製作者や使用を許諾されたユーザの情報が埋め込む
  - 音楽や画像などに対しての media watermarking は開発が進んでいる
  - もとのプログラムの観察的意味や実行効率を保ちつつ、簡単には改変されないような透かし情報を埋め込むための技術

演習 3 のまとめ (2)

- 1 週目 : Path-based watermarking
  - プログラムの条件分岐に基づいたフロー構造に透かしを埋め込む
- 2 週目 :Watermarking through register allocation
  - 変数の生存期間から作成された干渉グラフのカラーリング制約に透かしを埋め込む
- 3 週目 : Abstract watermarking
  - プログラムを抽象的に解釈 (ex. mod 空間) したときに抽出できるような透かしを埋め込む

演習 3 のまとめ (3)

- まだ実用に耐えるような技術は出現していないような印象を受ける
  - 実行効率と攻撃耐性のバランス
  - プログラムの意味を少したりとも変えてはいけないという厳しい制約 (cf. media watermarking)
- watermarking 単体ではなく、obfuscation や tamper-proofing との連携によって強度を上げていくのが現実解か
- アドホックな技術に加えて、数学の分野からのアプローチが待たれる

References

- Patrick Cousot and Radhia Cousot, "An Abstract Interpretation-Based Framework for Software Watermarking", In Conference Record of the 31st ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Programming Languages, Venice, Italy, January 14-16, 2004. ACM Press, New York, U.S.A. pp. 173—185.